Report No. UIUCDCS-R-80-1046 UILU-ENG 80 1746

THE PARAFRASE DATABASE USER'S MANUAL

BY
SHARON M. KUCK
DAVID A. MCNABB
STEPHEN V. RICE
YEHOSHUA SAGIV

DECEMBER 1980 CDC - I04AB

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-80-1046

THE PARAFRASE DATABASE USER'S MANUAL

BY

SHARON M. KUCK*
DAVID A. McNABB
STEPHEN V. RICE
YEHOSHUA SAGIV

DECEMBER 1980

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This research was supported in part by the Control Data Corporation under Grant No. CDC-I04AB.

*The work of this author was supported in part by the Department of Computer Science and the National Science Foundation under Grant No. US NSF MCS80-03308.

ABSTRACT

The PARAFRASE database is a relational database. It is implemented using an interface to an existing network database system that is available on the Prime. This manual describes Primos commands needed to execute the query processor, manipulate output files, and save/rerun query programs. The query language, consisting of an output specification and a Boolean expression, is described in detail. The relational view of the database for PARAFRASE is explained, including attributes, records, and the handling of null values. Finally, the control cards needed to store an Analyzer run in the database, and the update policy are described.

Key words : relational database, query language, null value

TABLE OF CONTENTS

I. INTRODUCTION

The Query Processor is designed to allow the user to retrieve information from his database by using a high-level relational model type query, though the underlying database is not of the relational type. The Query Processor (QP) reads in the user's query and produces a program, which is then run on the resident database system. The details of the QP implementation may be found elsewhere. This manual describes how to use the PARAFRASE database and the query language (i.e., how to formulate legal queries and specify options). Section IV describes the entry of Analyzer data into the database, and other information specific to the Analyzer. The reader is assumed to be familiar with the use of a terminal.

II. SYSTEM COMMANDS

This section contains a description and, in certain cases, an explanation of system commands that are necessary to use the Query Processor. First we will describe how to log on and off the terminal. Then we will give a description of the commands that are needed to execute the Query Processor. Finally, we will list a few system commands that are closely related to the usage of the Query Processor.

II.1 ACCESSING THE PRIME

To log on at the terminal type:

LOGIN <user id>

The system responds by typing:

PASSWORD:

At this point the user enters his password (which does not show up on the screen) followed by a carriage return.

To log off the system when the session is completed type:

LO

II.2 EXECUTING THE QUERY PROCESSOR

The user begins executing the Query Processor by typing in the proper system command as given in Figure II.1. In response, the Query Processor prompts the user for a query and then generates a query application program for the purpose of interrogating the database. Next the Query Processor executes the application program that prints the result of the query on the terminal screen or in a disk file as specified in the query. These steps are repeated until the user types a control-d followed by a carriage return instead of a query.

Before executing the application program, the Query Processor asks the user if the application program is to be saved. The user may answer yes, and give a name for the file for saving this application program. If the user saves the application program, he can subsequently execute it again without retyping the query or using the Query Processor (see explanation at the end of this section).

Figure II.1 contains a list of commands to execute the Query Processor. QP: and U: are not part of the commands but indicate what the Query Processor types and what the user types, respectively.

```
 U:  SEG PARA>#QPROC -A
QP:  Enter query after "Q?" prompt (or ctrl-d) to stop
QP:  Q?
 U:  <query>
QP:  Do you wish to save the query application program?
 U:  YES
QP:  Enter filename for application program:
 U:  <filename 1>
QP:  COMO -N

     The application program begins execution here.
     See Figure II.2.

QP:  COMO -N
QP:  SEG PARA>#QPROC -A
QP:  Enter query after "Q?" prompt (or ctrl-d) to stop
 U:  <ctrl-d>
QP:  Query processor terminated by user.
```

FIGURE II.1

Figure II.2 describes the messages printed on the terminal during the execution of the application program. The application program asks for the name of the file that the result of the query should be stored in if FILE or FILENOD was specified in the query (see Section III.1). If the file already exists, the system asks if it is OK to modify the file. If the user does not want the file to be modified (i.e.,the user answered "NO"), then the application program will ask for a different filename for the query result. Should the user have specified YES (i.e.,it is OK to modify the file), the user will have the option of overwriting the current file or appending the new result to it.

Figure II.2 gives a sequence of commands for execution of the application program. The application program (AP) is initiated by the Query Processor(QP).

```
AP:  ENTER FILENAME FOR QUERY RESULT:
 U:  <filename 3>
AP:  OK TO MODIFY OLD <filename 3>?
 U:  NO
AP:  ENTER FILENAME FOR QUERY RESULT:
 U:  <filename 4>
AP:  OK TO MODIFY OLD <filename 4>?
 U:  YES
AP:  OVERWRITE OR APPEND?
 U:  OVERWRITE
```

FIGURE II.2

If the application program is saved by the user as described earlier, subsequent executions of the application program are much faster. To initiate the applicaton program the user issues the PRIMOS command

SEG #<filename 1>

where <filename 1> is the name given by the user for storage of the application program (see FIGURE II.1). Execution of the application program then proceeds as stated in Figure II.2.

II.3 MISCELLANEOUS COMMANDS

II.3.1 LISTING A FILE

To obtain a line printer listing of the query result, which is stored in a file on disk, type:

SPOOL <filename> -FTN.

II.3.2 PROGRAM INTERRUPT

When the break key is hit during execution of the program type:

```
C ALL
CLUP
```

Failing to do this immediately after the break key was hit may cause errors later during execution of programs in the user space, because of open files, and/or may corrupt the database if the break key was hit during execution of a database retrieval program.

III. THE QUERY LANGUAGE

III.1 HOW TO SPECIFY WHERE YOUR OUTPUT GOES

III.1.1 THE DISPLAY COMMAND

The simplest place for the user to send his output is to the same terminal that he has used to enter his query. In this case one begins the query with the command "DISPLAY" (yes, in capital letters). This command is followed by the output list (see III) and tells the QP to send all the items in the output list to the terminal. No copy of this information is saved, or printed, so the user must be ready to read the screen or the user will miss his answers. [NOTE: To freeze the output on the screen, hold down the "CTRL" key while typing "S". To continue viewing, hold down the "CTRL" key and type a "Q". The output will then continue to scroll up the screen.] For example, if we wanted to see "TITLES" we would enter:

DISPLAY TITLES ...

The ... is the rest of the query as described in III and IV below.

III.1.2 THE FILE COMMAND

The user who wishes to save a copy of his output should use the "FILE" command. This stores the output in a file which may be retrieved at any time to be seen at the screen or printed. The terminal screen will still display the output as was the case above (III.1.1) except that there may be some changes in the number of columns, etc. For the same example above we enter:

FILE TITLES ....

Again, the .... is where the rest of the query goes (see III.2,III.3).

III.1.3 THE FILENOD COMMAND

The experienced user who doesn't need to see his output on the screen, or who wishes to enter another query with a minimum of delay, may choose the "FILENOD" command. As in the FILE command, a copy of the query output is stored in a file. In contrast to the other output commands however, FILENOD will not display any output at the user's terminal. When the query has been processed, and the output has been put in the file, the QP will simply ask for the next query. The user must then take care of any printing, listing, etc as for any other diskfile. As one might expect, our example now becomes:

```
FILENOD TITLES   ...
```

[NOTE : The command FILENOD means file with no display.]

III.2. HOW TO SPECIFY WHAT OUTPUT YOU GET.

III.2.1 THE OUTPUT LIST

The output list consists of a list of names and function calls. The next section discusses function calls in detail. (III.2.2)

There are two types of names that may be placed in the output list: attribute names and record names. The user must be familiar with these names to ask even a simple query. (If you do not know these names, see your database administrator.)

Attribute names are names given to specific items of information in the database. The name gives a clue as to what the information means. For example, a library database might use AUTHOR and TITLE as attribute names. Their meanings are obvious. A query beginning:

```
DISPLAY AUTHOR ...
```

would display name(s) of author(s) on the screen. Which names were displayed would depend on the ... part of the query (see section III.3).

Often there are groups of logically related attribute names. In the previous library example, the attribute names AUTHOR, TITLE, DATE, PUBLISHER might all be grouped together into the record BOOK. If the user would like information from all the attributes in one group he could just give the record name for the group as an item in the output list. This will have the same effect as typing each of the attribute names. For example, the following queries are all equivalent (assuming the ... parts are the same as well):

```
DISPLAY AUTHOR, TITLE, DATE, PUBLISHER ...

DISPLAY BOOK ...
```

The first query illustrates the last point about output lists, that each item (record name, attribute name, or function call) should be

delimited from the next by at least one blank and/or comma. The following queries are all equivalent and legal.

```
DISPLAY AUTHOR TITLE DATE   ...

DISPLAY,AUTHOR,TITLE,DATE   ...

DISPLAY, , ,AUTHOR,,,TITLE   DATE   ...

, , DISPLAY      AUTHOR,,,,TITLE DATE   ...
```

Blanks and commas are used as delimiters throughout the query. They are ignored except that they show where one name ends and the next name begins.

III.2.2 BUILT-IN FUNCTIONS

There may be cases in which a user is less interested in a list of values for some attribute than in having one aggregate value which is a function of the values in the list. If that function is available as a built-in function, the computation can be performed by placing a function call in the output list. At present, the only built-in function is the function "average of one or more attribute or record names" (AVG). If the database contained student information, such as the attributes AGE and EXAMSCORE, the following query would give a list of ages and a list of scores.

```
DISPLAY  EXAMSCORE  AGE  ...
```

The user could then average each list by hand. These two averages could also be obtained by entering:

```
DISPLAY AVG(EXAMSCORE, AGE)  ...
```

One average will be output for each unique attribute name appearing in the function call. [NOTE: Only numerical-type attributes can be averaged.] Record names may also appear in the AVG function call, but only those records for which special averaging routines have been devised. [Note: In the PARAFRASE database, only stat records can be averaged.] Duplicate names in AVG calls (same or succeeding calls)

will be ignored. The following queries will have identical output:

```
DISPLAY AVG(AGE,EXAMSCORE)   ...

DISPLAY AVG(AGE) AVG(EXAMSCORE)   ...

DISPLAY AVG(AGE,AGE,EXAMSCORE) AVG(EXAMSCORE)   ...
```

At present, no other special functions have been implemented.

III.3. HOW TO ASK QUESTIONS ABOUT THE DATABASE

III.3.1 THE "WHERE" COMMAND

All the preceeding examples contained ... to signify the rest of the user's query. If the user wants no restrictions on which tuples are included in the output, this ... should be replaced by a semicolon ";". This non-restricted query is said to contain a "null where clause". (The WHERE command is not used at all.) For example, these two (legal and complete) queries contain null where clauses:

```
DISPLAY TITLE;

FILE AVG(AGE) EXAMSCORE ;
```

[NOTE: If the semicolon is omitted, the QP will prompt the user for the next line of his query and just wait.]

In order to ask for an output list whose members have some property (other than that they are in the database) one must specify the conditions that must hold before a value is output. As in the previous library example, suppose we want a list of all titles of books written by the author 'KNUTH'. The correct query would then be:

```
DISPLAY TITLE WHERE AUTHOR = 'KNUTH';
```

The "WHERE" command restricts the output to titles of books whose corresponding AUTHOR attribute is a string of characters equal to 'KNUTH'. The semicolon marks the end of the query. (If you forget the semicolon, the QP will continue to wait for more lines of the query to be entered. Very long queries should be entered one line (<120 chars) at a time, each line followed by a RETURN keystroke.)

III.3.2 LOGICAL EXPRESSIONS - THE "WHERE" CLAUSE

The WHERE command and everything to it's right (up to the semicolon) is called the where clause. The where clause is actually one logical expression (boolean expression). The query processor

evaluates the expression for each tuple in the database, and if the expression is true, the output requested (sections III.1,III.2 above) occurs for that tuple. In the example above, each tuple with AUTHOR = 'KNUTH' will give a TRUE value to the where clause and cause the corresponding value of AUTHOR to be displayed. If there were 3 such tuples in the database, the query:

```
DISPLAY AUTHOR WHERE AUTHOR = 'KNUTH';
```

would produce the following list as output:

```
KNUTH
KNUTH
KNUTH
```

This query is an example of the comparison of a string-constant with a string-valued attribute. Boolean-valued attributes can be compared with either of the two boolean constants TRUE or FALSE. Numerical-valued attributes can appear in arbitrary arithmetic expressions which can then be compared to other arithmetic expressions. Unlike string and boolean attributes, arithmetic comparisons can use all six relational operators, not just the equal operator. (see III.3.3)

The expression AUTHOR = 'KNUTH' could be combined with other simple expressions to produce larger logical expressions. For example the following query asks for titles of books written by either Knuth or Dijkstra:

```
DISPLAY TITLE WHERE [AUTHOR = 'KNUTH' OR AUTHOR = 'DIJKSTRA'];
```

Note the use of the square brackets. This is required around the entire where clause only when the expression is not in conjunctive form, that is, whenever the top-level operator(s) are not all AND. The possible logical (boolean) operators are AND, OR, and NOT. Normal precedence rules apply. (NOT > AND > OR). Square brackets must be used to override precedence.

Suppose we want titles of all books written by Knuth and cost below 20. The following query expresses this:

```
DISPLAY TITLE WHERE AUTHOR = 'KNUTH' AND COST < 20;
```

Square brackets were not required here because only AND appears at the top level of the logical expression (WHERE ____ AND ____ ;). Extra square brackets can appear around any logical subexpression without changing the result (as long as precedence is not changed):

```
DISPLAY TITLE WHERE [[ AUTHOR = 'KNUTH'] AND [ COST < 20 ]];
```

Anything appearing in square brackets must be able to return a true or a false value, and each pair of square brackets must contain at least one relational operator:

```
DISPLAY TITLE WHERE [ AUTHOR = 'KNUTH' ];      (<- correct)

DISPLAY TITLE WHERE [AUTHOR] = 'KNUTH' ;       (<- incorrect)
```

Some of these restrictions are only temporary. For more details, see section III.3.4.

III.3.3 ARITHMETIC EXPRESSIONS

Any numerical-valued attribute or constant can appear in an arithmetic expression. An example was given above by the query fragment COST < 20. The arithmetic expression COST is compared to the arithmetic expression 20 to give a logical result (true or false). Much more complicated expressions are allowed using the operators {+,-,*,/,**}. For example:

```
DISPLAY TITLE WHERE COST < 126 - (15 + 6)/7;
```

Any number of arithmetic-valued attributes or constants may be combined into one arithmetic expression:

```
DISPLAY TITLE WHERE COST < (INCOME - TAX)/12 + BONUS ;
```

Note the use of parenthesis to override the precedence of the arithmetic operator / over the arithmetic operator - . Every expression appearing inside matching pairs of parenthesis must return an arithmetic value. [NOTE: A string constant, a logical constant,

or an attribute may also be surrounded by matching parentheses. This will not cause an error but is always unnecessary. See III.3.4]

Two arithmetic expressions can be combined into a logical expression by any of the operators {<,>,=,<=,>=,<>} which are:

<	(less than)
>	(greater than)
=	(equal)
<=	(less than or equal to)
>=	(greater than or equal to)
<>	(not equal to)

None of these operators can ever appear inside a matching pair of parentheses since they return logical values (boolean)! The following query illustrates the legal use of brackets and parentheses:

```
DISPLAY TITLE WHERE [[((COST)) >= (.15)*(SALARY-7)]];
```

III.3.4 SPECIAL RESTRICTIONS

In addition to the square-bracket restriction mentioned in III.3.2, the following also apply. (Some are temporary.)

1. An attribute that has a character string as a value can be compared only to string constants with length less than or equal to the maximum possible string-length of that attribute.
2. Logical attributes can be compared ONLY to logical constants.
3. Null values for numerical attributes are stored as large negative numbers, and thus a selection such as COST < 20 is true even if cost is null. The user must explicitly exclude nulls if this is desired.
   (i.e. COST < 20 will be true for all null values stored for COST. Explicit exclusion of nulls would be realized by COST < 20 AND COST >= 0.)

III.4. ERROR MESSAGES

There are three major types of error messages produced during parsing of a user's query: recoverable errors (warnings), unrecoverable errors, and internal errors.

Internal errors are always marked INTERNAL ERROR and occur on overflow of internal tables, on discovery of invalid data in schema or name tables, etc. They indicate that either the query was too long or there is a bug in the system. Internal errors should be reported to the database administrator at once.

Recoverable errors or warnings are given for such things as appending extra characters on six-character attribute names (six is the maximum length for attribute names), typing unrecognized characters, etc. The user should be sure that he understands the reason for the warning.

Unrecoverable errors include unrecognizable names, incompatable types in the same expression, operand-operator incompatablity (such as strings and the < operator), etc. These errors cause rejection of the entire query and reinitialization of all internal structures. The query processor will then prompt for a new or revised query. (If you do not understand why your query was rejected, see your database administrator.)

IV. THE PARAFRASE DATABASE

IV.1 ANALYZER CONTROL CARDS

To store an Analyzer run in the database, the following control cards must be used:

```
/*ID PUNCH=CYBER,CARDS=10000,NAME='file(user number)'
  .
  .
  .
// EXEC ANALYZE,RUN=STATISTICS
  .
  .
  .
//INPUT DD *
  .
  .
  .
&NAME=user's last name
&TYPE=machine type name
&DEFINITION=definition number
&DESCRIPTION=machine type description
```

After the run is finished, type the following:

```
GET,file/UN=usernumber

APPEND,DBASE,file/UN=3PARAFR
```

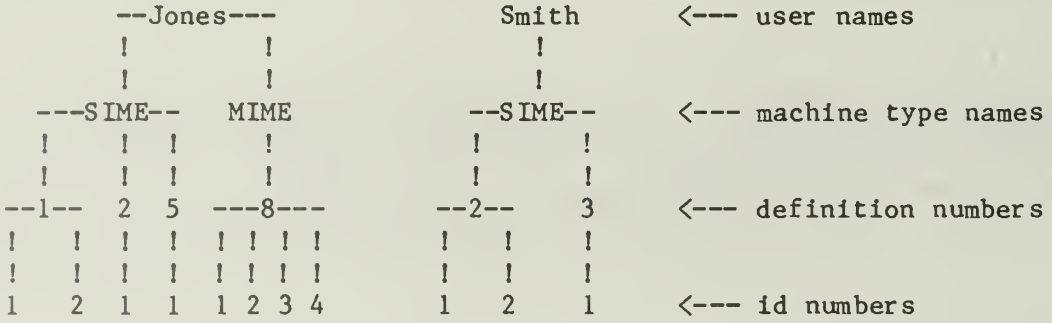
Note: In each of the first three "&-cards", blanks are not allowed.

IV.2. MACHINE TYPES

A machine type is an entity used to describe a type of computer architecture. It is uniquely defined by a user name, a machine type name, a definition number and an id number. Associated with a machine type are the names and versions of the passes and an 80-character machine type description. The user name, machine type name, definition number and machine type description are supplied by the user via control cards submitted in an Analyzer run. An id number is assigned by the update program before insertion into the database.

The assignment of an id number is done to differentiate machine types that have the same user name, machine type name and definition number, but have a different set of passes and/or versions. The id number assigned to the first unique set of passes/versions is 1, the second unique set of passes/versions is 2, etc., for a given user name, machine type name and definition number.

Example of a hierarchy of machine types in the database:



In a query, one may use whatever portion of the trees that he desires. By specifying a user name, a machine type name, a definition number and an id number, a single machine type is specified (i.e., one of the leaves). By not specifying the id number, one will get all machine types having the specified user name, machine type name and definition number, regardless of id number. (This may be desirable if differences in passes/versions have not altered the semantics of the machine type.)

Similarly, one can specify only a user name and a machine type name, if the definition is not of concern. If one wants all machine types he's defined to be used in a query, he need only specify his

user name. If one wants everybody's machine types to be taken into consideration in a query, a value for none of the four attributes is specified.

IV.3. NULL VALUES IN THE OUTPUT

Any field containing all asterisks in the output indicates that no value was stored for this attribute, i.e. the attribute contains a "null value." In the output of the average of an attribute or a record, an occurrence of all asterisks in a field indicates that each tuple retrieved contained a null value for this attribute. Averages are computed on non-null values only - null values are never averaged in with non-null values.

IV.4. UPDATE POLICY

An Analyzer run consists of a machine type, a program, a set of options, the date and time of the run, and various statistics computed. A machine type, a program and a set of options uniquely determine an Analyzer run. Therefore, before an Analyzer run is inserted into the database, a check is made to determine whether there already exists a run in the database having the same machine type, program and set of options. If so, this run is deleted before the new run is inserted.

IV.5. ATTRIBUTES IN THE PARAFRASE DATABASE

1. Machine Types — record name = MTYPES

1. name of user (character(10))	USER
2. name of machine type (character(10))	MT
3. definition number of machine type (integer)	DEF
4. id number of machine type (integer)	ID
5. description of machine type (character(80))	INFO
6. pass names (character(8) for each pass)	******
7. pass versions (character(10) for each pass)	******

(****** means attributes may not be specified explicitly in a query)

2. Programs — record name = PROGS

1. package name (character(8))	PKG
2. program name (character(8))	PRG

3. Options — record name = OPTNS

1. no-side-effects (boolean)	NOSDEF
2. pipeline-dd (boolean)	PIPEDD
3. vector-reg (boolean)	VECTRG
4. do-bound (integer)	DOBND
5. reg-length (integer)	REGLEN

4. Run-date — record name = RUNS

1. date of run (character(10))	DATE
2. time of run (character(8))	TIME

5. Compiler Statistics 1 (all integer) — record name = CS1

1. Source Statistics	
1. cards	CARDS
2. statements	STMTS
3. do-loops	LOOPS
4. maximum do-loop nest	NEST

2. If-Loop Translation	
1. if-loops transformed to while-loops	IFTWL
2. if-loops transformed to do-loops	IFLOOP
3. Loop Freezing	
1. loops manually frozen	HNDFRZ
2. loops auto-frozen with calls	CALFRZ
3. loops auto-frozen with if-loops	IFLFRZ
4. loops auto-frozen with io-statements	IOFRZ
5. loops auto-frozen with loops exits	EXTFRZ
6. loops auto-frozen with branches around loops	SKPFRZ
7. loops auto-frozen for other reasons	OTHFRZ
8. total number of loops auto-frozen	AUTFRZ
9. total number of loops frozen	FROZEN
4. Scalar Renaming	
1. variables renamed	RENAME
2. new variables created	NEWNAM
5. Scalar Invariant Code Floating	
1. expressions floated	EXPFLT
6. Induction Variables	
1. total number of induction variables	INDUCT
2. induction variables of nest i, i=1,2,3	IVNLi
3. induction variables of nest > 3	IVNLG3
4. induction variables with i increments, i=1,2,3	IViINC
5. induction variables with > 3 increments	IVG3IN
7. Scalar Expansion	
1. total number of scalars expanded	EXP
2. scalars expanded by i dimensions, i=1,2,...,7	DIMi
3. scalars expanded by > 7 dimensions	DIMG7
8. Dead Code Elimination	
1. statements eliminated by general dead code	DEAD
2. statements eliminated by scalar dead code	DEADSC
9. Loop Collapsing	
1. loops collapsed	COLLAP
10. Loop Fusion	
1. loops fused	FUSE
11. Loop Interchanging	
1. loops interchanged for strip mining	INTSM
2. vector loops interchanged	INTVEC
3. vector loops found	VLFND

6. Compiler Statistics 2 (all integer) record name = CS2

1. Statement Function Expansion
   1. statement functions — STMTFN
   2. statement function substitutions — SUBST

2. Vector FORTRAN Translation
   1. vector statements found — VSTFND
   2. loops added — LPSADD

3. Loop Blocking
   1. block size — LBLKSZ

4. Compiler Temporary Reusing
   1. variables deleted — VARDEL
   2. temporaries left — TMPLFT

5. Compiler Temporary Shrinking
   1. compiler arrays shrunk — ARRSHR

6. Define Variables
   1. variables defined — VARDEF

7. Wrap Around Variables
   1. wrap around variables — WAV
   2. loops — WAVLPS

8. Carry Around Variables
   1. carry around variables — CAV
   2. loops — CAVLPS

9. Scalar Forward Substitution
   1. scalars forward substituted — FRWRD
   2. substitutions made — SFSUBS

10. LHS Code Floating
    1. statements floated — LHSFLT
    2. loops deleted — LHSLDL

11. RHS Code Floating
    1. statements floated — RHSFLT
    2. loops deleted — RHSLDL

12. "Code" Generation
    1. vector statements — VEC
    2. scalar statements — SCAL
    3. recurrences — REC
    4. vector-recurrences — VECREC
    5. io-statements — IO
    6. conditionals — CONDS
    7. serial loops — SERLLP

8. calls CALLS

7. Recurrence and If-Removal Statistics (all integer)
record name = RECURS

1. Recurrence Statistics
   1. recurrences translated RECS
   2. vector sums
      1. total number of vector sums VSUMT
      2. number of full vector sums VSUMF
      3. number of modified vector sums VSUMM
      4. number of anti-modified vector sums VSUMA
   3. vector products VPRODx
   4. dot products DPRODx
   5. R<n,1> RN1x
   6. R<n,1> CC RN1CCx
   7. R<n,1> RT RN1RTx
   8. R<n,1> CC RT RCCRTx
   9. vector MAX VMAXx
   10. vector MIN VMINx
   11. vector ALL VALLx
   12. vector ANY VANYx

   where x=T,F,M,A

2. IF-Removal
   1. IFs changed to MAX or MIN MAXMIN
   2. IFs in loops removed IFS
   3. modes used MODES
   4. exit modes EXTMDS
   5. mode modifications MODEMD
   6. loop invariant modes floated LIMFLT
   7. leading zeroes found (exit-IFs) LZFND

8. Loop Structure (all integer) record name = LOOPST

1. Vectorized Loop Structure - 10 X 10 table VLij

2. Inverted Vectorized Loop Structure - 10 X 10 table
INVLij
where i,j=S,0,1,2,3,4,5,6,7,G
(S means "sum" and G means ">7")

9. Sime Simulation Statistics 1 (all real) record name = SIME

1. T(p), p=2**n, n=0,1,...,15 Tp

2. O(p), p=2**n, n=0,1,...,15 Op

3. Serial Statistics
   1. serial Tl SERLTl
   2. S(p), p=2**n, n=0,1,...,15 Sp
   3. E(p), p=2**n, n=0,1,...,15 Ep
   4. R(p), p=2**n, n=0,1,...,15 Rp
   5. U(p), p=2**n, n=0,1,...,15 Up
   6. Q(p), p=2**n, n=0,1,...,15 Qp
   7. Alpha(p), p=2**n, n=0,1,...,15 Ap
   8. Max S(p) MAXS
   9. Max E(p) MAXE
   10. Max R(p) MAXR
   11. Max U(p) MAXU
   12. Max Q(p) MAXQ
   13. Max Alpha(p) MAXA

4. Parallel Statistics
   1. parallel Tl PARLTl
   2. S(p), p=2**n, n=0,1,...,15 SPp
   3. E(p), p=2**n, n=0,1,...,15 EPp
   4. R(p), p=2**n, n=0,1,...,15 RPp
   5. U(p), p=2**n, n=0,1,...,15 UPp
   6. Q(p), p=2**n, n=0,1,...,15 QPp
   7. Alpha(p), p=2**n, n=0,1,...,15 APp
   8. Max S(p) MAXSP
   9. Max E(p) MAXEP
   10. Max R(p) MAXRP
   11. Max U(p) MAXUP
   12. Max Q(p) MAXQP
   13. Max Alpha(p) MAXAP
   14. P with maximum Qp PMAXQ

10. Sime Simulation Statistics 2 (all real) record name = SIMEH

1. Utilization(p,1), p=2**n, n=0,1,...,15, 1=2**m, m=0,1,...,n
   UTnm

2. Vector-Length(p,1), p=2**n, n=0,1,...,15, 1=2**m, m=0,1,...,n
   Vnm

3. Sigma(p), p=2**n, n=0,1,...,15 Zp

11. Vector Register Statistics (all integer) record name = VRSTAT

1. Vector Register Assignment

1. vector registers allowed — VR
2. mode registers allowed — MR
3. register length — REGL
4. vector registers used — VRUSED
5. mode registers used — MRUSED

2. Register Chaining
   1. register chains found — CHAIN
   2. statements deleted — DEADCH

3. Pipeline Filling
   1. original pipelining factor — ORIGPF
   2. final pipelining factor — FINLPF
   3. pipelining improvement — PIPFIL

12. Memory Hierarchy Statistics (all integer) record name = MEMHST

1. Clustering
   1. Before Transformation
      1. name partitions — BNP
      2. minimum number of statements per np — BMINST
      3. maximum number of statements per np — BMAXST
      4. average number of statements per np — BAVGST
   2. After Transformation
      1. name partitions — ANP
      2. minimum number of statements per np — AMINST
      3. maximum number of statements per np — AMAXST
      4. average number of statements per np — AAVGST

2. Loop Fusion
   1. name partitions fused — NPFUSE
   2. name partitions — NP

3. Block Indexing
   1. name partitions block indexed — BLKNDX

4. Non-Basic to Basic
   1. non-basic name partitions — NONBAS
   2. non-basic name partitions changed to basic name partitions — BASIC

5. Scalar Transformations
   1. scalars forward substituted — SCLFRW
   2. scalars expanded — SCLEXP

13. Trace Generator Statistics record name = TRCGEN

1. Data Input Statistics
   1. page size (integer) PGSIZE
   2. name of input variable i, i=1,...,10 (character(7)) ******
   3. value of input variable i, i=1,...,10 (character(10)) ******

(****** means attributes may not be specified explicitly in a query)

2. Simulation Statistics For Each Memory Allocation (all integer)
   1. page fault i, i=1,2,...,100 PGFi
   2. space time cost i, i=1,2,...,100 STCi

3. Overview Statistics (all integer)
   1. array references ARRREF
   2. virtual pages VIRPGS
   3. minimum page fault MINFLT
   4. memory allocation of minimum page fault MEMFLT
   5. minimum space time cost MINSTC
   6. memory allocation of minimum space time cost MEMSTC
   7. declared arrays NARRAY

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-80-1046	2.	3. Recipient's Accession No.
4. Title and Subtitle THE PARAFRASE DATABASE USER'S MANUAL			5. Report Date December 1980 6.
7. Author(s) Sharon M. Kuck, David A. McNabb, Stephen V. Rice, Yehoshua Sagiv			8. Performing Organization Rept. No. UIUCDCS-R-80-1046
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. CDC-I04AB
12. Sponsoring Organization Name and Address Control Data Corporation Minneapolis, Minnesota			13. Type of Report & Period Covered Technical Report 14.

15. Supplementary Notes

16. Abstracts

The PARAFRASE database is a relational database. It is implemented using an interface to an existing network database system that is available on the Prime. This manual describes Primos commands needed to execute the query processor, manipulate output files, and save/rerun query programs. The query language, consisting of an output specification and a Boolean expression, is described in detail. The relational view of the database for PARAFRASE is explained, including attributes, records, and the handling of null values. Finally, the control cards needed to store an Analyzer run in the database, and the update policy are described.

17. Key Words and Document Analysis. 17a. Descriptors

relational database
query language
null value

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
RELEASE UNLIMITED	UNCLASSIFIED	30
	20. Security Class (This Page) UNCLASSIFIED	22. Price